Resolution of the big-data problem related to a dimension reduction algorithm based on Multi-Scale similarities in Stochastic Neighbor Embedding

Author:

Fabian Souris

souris.fabian@gmail.com

August 17, 2015

Supervisors:
Prof. John A. Lee
Prof. Michel Verleysen

Acknowledgments

First of all, I would like to give a special thanks to my co-promoter, John Lee, for all these really interesting talks and his contribution to this work. He taught me to always think positively because, in the research field, everything is possible.

My sincere thanks also goes to my promoter, Prof. Verleysen, and the rest of my master thesis committee: Mr. Paul and Mrs. Bunte, for their time dedicated to my work.

Thanks to Thibault Sottiaux, Charles Thomas and Kevin Souris for their content reviews of my work. And thanks to Pauline Romnee and Noemie Thomas for their orthographic reviews.

Last but not least, I would like to thank my family and my friends, especially Justin Loroy and Ionel Munteanu, for supporting me spiritually throughout writing this master thesis.

Abstract

Data visualization has always been a necessity. That is why the dimension reduction field is an important part of machine learning. One of the best algorithms to do data visualization is the multi-scale stochastic neighbor embedding (Ms. SNE). But because of its time complexity of $O(N^2 \log(N))$, it is not suitable for large databases. In order to solve this Big Data problem, the solution proposed here is an accelerated version of Ms. SNE. It uses metric trees to approximate the data cloud into clusters and to reduce the cost to a $O(N \log^2(N))$ time complexity. This is a new research and the resulted solution is not perfect yet but the results prove that the approximations added to the original algorithm allow the code to run on larger databases with a minimum loss of precision.

Keywords— Nonlinear dimensionality reduction, Big Data, Data visualization, Stochastic neighbor embedding, Multi scale SNE, Metric trees

Contents

0.1 Introduction

The structure of the data we work with is always useful to know. Unfortunately, the human brain can only think and see data in at most three dimensions.

It is exactly the aim of the DR (Dimension Reduction) algorithms. It is an old field of research, starting with some linear solutions (like Principal Component Analysis) which basically search for the best combination of dimension to keep as much information about the data as possible. We have now more complex solutions that use non-linear methods to keep even more information. In this paper we will work on the Multi-scale Stochastic Neighbor Embedding method which tries to keep the same neighborhood for all points between the high dimensional space and the low dimensional space.

As an example, suppose we have a postal service which want to automate the mail routing based on the postal code. It has thousands of envelopes, letters and packages. Its aim is to read the handwritten postal codes and sort all items. To learn its algorithm, it has a huge database of handwritten digits, the MNIST database [11]. It is about 70000 images ($20 * 20px$). And the company want to visualize these data.

The first problem of the MNIST database is its $20 * 20 = 400$ dimensions. You can not see the global topology with the original data set since you can not show on a graph all dimensions at once. The DR will reduce the number of dimension to 2 or 3 and keep the most information as possible.

The second problem is the number of data (70000 images). Nowadays information technology permits us to record everything and create huge databases. But some of our old information retrieval methods become obsolete in these conditions. We call that problem the Big Data problem. It is the case in the DR field which generally demands that we compare each pair of data. With N data, we have a time complexity of at least $O(N^2)$ which is inconceivable with that size of data sets.

The solution to the Big Data problem is generally to approximate the algorithm solution with some "shortcuts". In our case, we will try to compare each point with clusters of points. The original Ms. SNE (Multi-scale Stochastic Neighbor Embedding) time complexity is $O(N^2 \log(N))$. We will see how to reduce that to $O(N \log^2(N))$ using approximation techniques.

Firstly, we will see a brief piece of the DR history. Next we will explain the original algorithm that will be improved and its family in the DR field. Then we will introduce the Big Data problem, some existing solutions and basics techniques usually used in this case. After that theoretical background, we will explain our solution for the MS. SNE algorithm, the challenges encountered and the solutions founded. Finally you will find the results of the new algorithm, we will analyze and discuss them.

Chapter 1

Dimension reduction history

Before get in the details of the accelerated algorithm, let us first have a look at the already existing ones in the field of dimension reduction. We will see in this chapter all principal algorithms from the oldest one (principal component analysis) to the recent non-linear methods [17].

The DR field maps data in HD (high dimensional) space into a LD (low dimensional) space. We define a HD distribution of N points as $\mathbf{\Xi} = [\boldsymbol{\xi}_i]_{1 \leq i \leq N}$ and the resulted LD distribution as $\mathbf{X} = [\mathbf{x}_i]_{1 \leq i \leq N}$. These algorithms are generally based on distance between points (Not necessarily with the Euclidean metric). We note the HD distance between $\boldsymbol{\xi}_i$ and $\boldsymbol{\xi}_j$ as δ_{ij}. Similarly, we define the LD distance between $\mathbf{x}_i$ and $\mathbf{x}_j$ as d_{ij}.

1.1 Linear dimensionality reduction

PCA (Principal component analysis) (1901) [20] and CMMDS (Classical metric multidimensional scaling) (1938) [29] are both linear DR algorithms. They take the eigenvalue decomposition of two matrix, covariance matrix ($\Xi\Xi^T/N$) for PCA and Gram matrix ($\Xi^T\Xi$) for CMMDS.

For the PCA, the decomposition gives us a new orthogonal basis for the data. The first axis is the direction of the maximum variance in the data. The dimension reduction only take the N most important axis and drop the others.

The MDS reduces the dimensionality while preserving the inner products between the data point [8]. The eigenvalues obtained are the same as the PCA up to a constant. Both functions return the same result.

Their time complexity is divided in two parts. The matrix computation is done in $O(P^2N)$, the eigen-value decomposition is in $O(P^3)$. The global time complexity is $O(P^2N + P^3)$, where P the number of dimensions in the initial space.

The PCA will be used for initialization in our algorithm.

1.2 Stress-based MDS and Sammon mapping

Stress-based MDS [7] and Sammon mapping [21] (1952 and 1962 respectively) are based on an objective function minimization. These functions are based on the Euclidean metric for the LD distances d_{ij} but not necessarily for the HD distances δ_{ij}.

The objective function of the Stress-based MDS (multidimensional scaling) is

$$E(X, \Delta, W) = \frac{1}{C} \sum_{i,j=1}^{N} w_{ij}(\delta_{ij} - d_{ij})^2 \ , \tag{1.1}$$

where Δ is a symmetric distance matrix, X is the set of N points and W is a matrix of weights used to minimize E in a first part [5]. The first step is to find $\hat{W}$ that minimizes

$$E(X, \Delta, \hat{W}) = \min_{W} E(X, \Delta, W) \ , \tag{1.2}$$

then it finds optimal $\hat{X}$ by

$$E(\hat{X}, \Delta, \hat{W}) = \min_{X} E(X, \Delta, \hat{W}) \ . \tag{1.3}$$

The Sammon mapping cost function is

$$E(X, \Delta) = \frac{1}{\sum_{i,j=1}^{N} \delta_{ij}} \sum_{i,j=1}^{N} \frac{(\delta_{ij} - d_{ij})^2}{\delta_{ij}} \ . \tag{1.4}$$

It minimizes directly X.

The time complexity for these algorithm is $O(N^2)$ because of the needs to compare each pair of points.

1.3 Self-organizing map

The SOM (Self-Organizing Map) (1982) [9, 14]is a nonlinear version of PCA. Based on an articulated grid, it tries to fit the grid trough the data cloud. Each node of the grid is associated with a weight vector in HD and a position in LD. At each iteration of the algorithm, each node of the grid will be updated to a position closer to the set of points that are closest to that node. At the end of the iterative process, the method maps the point on the LD grid to generate the LD visualization. The grid has to be initialized. The easy solution is to initialize the grid randomly in the data cloud. A smarter technique is to set the grid using a PCA to be closer to the final solution. You can see an example of a SOM application in Figure 1.1. It is a very simple nonlinear algorithm which is surprisingly efficient to unfold some data topologies (global structures).

For some data topologies, this method is not recommended. The shape of the grid must be predefined and does not depend on data. If the topology is a complex structure, this solution is not a good one.

1.4 Nonlinear spectral methods

Nonlinear spectral methods are basically the same as PCA or CMMDS but with a preprocessing of the data with a nonlinear transformation. The optimization

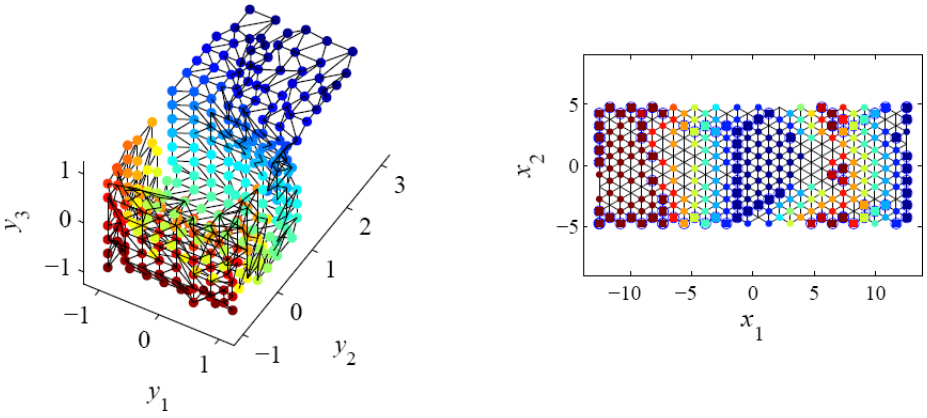


Figure 1.1: Example of SOM application to a 3D open cube

remains convex but the nonlinear transformation is not optimized, except in a few cases. These transformations might remove some topological constraints and the time complexity remains $O(n^2)$ as for the classical PCA.

1.4.1 Isomap

The idea behind isomap (1998) [22] idea is to replace the classical metric with a smart one. Instead of distances like the Euclidean one, it computes the shortest paths in a graph of K-ary neighborhoods for distant points. As we can see on

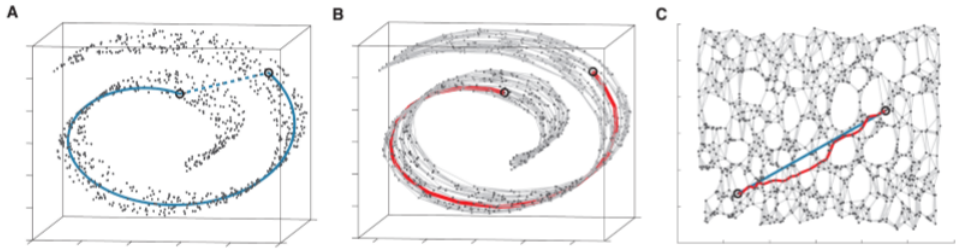


Figure 1.2: Illustration of the distance computation in the Isomap case on folded data

Figure 1.2, it is effective to unfold data.

1.4.2 Laplacian eigenmaps and Maximum variance unfolding

The basic idea of Laplacian eigenmaps (2002) [2] is to shrink distances between neighbors and avoid trivial solutions and undeterminacies. This method mini-

mizes distances between projections in the LD space, it minimizes

$$E_{LE} = \sum_{i,j=1}^{N} \|x_i - x_j\|^2 w_{ij} \ , \tag{1.5}$$

$$\text{with } w_{ij} = \exp\left(-\frac{\|y_i - y_j\|^2}{2T^2}\right) \tag{1.6}$$

where T is a parameter chosen by the user..

It is the opposite for Maximum variance unfolding (MVU) (2004) [28]. It tries to stretch distances of non-neighboring points.

Chapter 2

Stochastic Neighbor Embedding and Multi-scale approach

Stochastic neighbor embedding (SNE) methods are the most recent in this field. SNE warps distances into similarities with specific properties, which allow this method to defeat some problems related to the dimensionality introduced hereafter. It try to keep the same neighborhood for each point in HD and LD. The global structure of these algorithms is to minimize a cost function with an optimization technique.

2.1 Curse of dimensionality

Since the aim is to map high-dimensional data into a low-dimensional space, we have to care of the norm concentration of all pair of points [16]. As you can see in Figure 2.1, the higher the number of dimensions M is, the most offset the mean of the norm concentration have. These norm concentrations represent a property of such a dimensional space. If we try to visualize a HD data set in a LD space without regarding that property (using only Euclidean distances for example), we will have a mismatch. Thus a shift is needed to correspond HD norm concentration and LD norm concentration.

A solution is to normalize the metric used to make it independant of the number of dimensions. Most of the Multidimensional scale methods compare HD distances and LD distances without considering this shift. That is why they suffer from the curse of dimensionality.

2.2 Similarity

The similarity is the metric used by SNE-like methods. As we said (Section 2.1), that metric is normalized and is independent of the number of dimensions of the space.

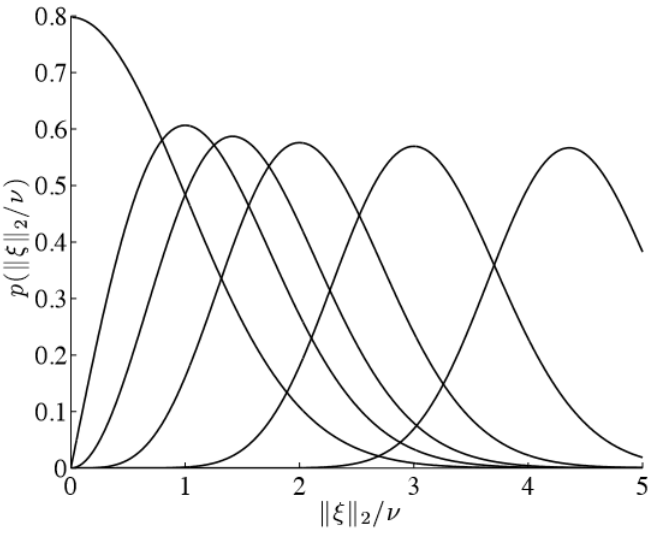


Figure 2.1: Pairwise distance probability density function for $M \in \{1, 2, 3, 5, 10, 20\}$

2.2.1 Iso-entropic similarities

We have a set of N points $\mathbf{\Xi} = [\boldsymbol{\xi}_i]_{1 \leq i \leq N}$ in a M-dimensional space and its representation in a P-dimensional space $\mathbf{X} = [\mathbf{x}_i]_{1 \leq i \leq N}$, with $P \leq M$. $\delta_{ij} = \|\boldsymbol{\xi}_i - \boldsymbol{\xi}_j\|_2^2$ and $d_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ denote the squared Euclidean distances between the ith and jth points in the HD and LD spaces, respectively. We also have $\pi_i = \lambda_i^{-2}$ and $p_i = l_i^{-2}$ that are the precisions of the normalized Gaussian functions centered on $\boldsymbol{\xi}_i$ and $\mathbf{x}_i$ with λ_i and l_i, the bandwidth of the Gaussian functions. From these notations, we define the similarities in the M- and P-dimensional spaces as

$$\sigma_{ij} = \frac{\exp(-\pi_i \delta_{ij}/2)}{\sum_{k,k \neq i} \exp(-\pi_i \delta_{ik}/2)} \quad \text{and} \quad s_{ij} = \frac{\exp(-p_i d_{ij}/2)}{\sum_{k,k \neq i} \exp(-p_i d_{ik}/2)} \ . \tag{2.1}$$

for $i \neq j$. $\sigma_{ij} = s_{ij} = 0$ for $i = j$ by convention. Similarities σ_{ij} and s_{ij} can be seen as a probabilistic membership to soft Gaussian neighborhoods.

Now we need to fix the size of the neighborhoods. But what is the size in such a definition? We can define a perplexity K which is a relationship between the bandwidth and a generalized concept of size. K_i depends on the entropy H_i of σ_{ij}. We have

$$K_i = \exp(H_i) \quad \text{and} \quad H_i = -\sum_{j=1}^{N} \sigma_{ij} \ln \sigma_{ij} \ . \tag{2.2}$$

In all SNE algorithms, the user can choose a perplexity $K_\star$ to adjust the size of the neighborhood and the algorithms solve $\ln(K_\star) = H_i$ for $1 \leq i \leq N$ to get the precision π_i. Thus σ_{ij} are iso-entropic. For the LD space, SNE set $p_i = 1$ for all i and NeRV (Neighborhood Retrieval Visualizer) do $p_i = \pi_i$.

2.3 Cost function

Again, the similarities σ_{ij} and s_{ij} can be seen as probability. Since they have the property $\sum_j \sigma_{ij} = \sum_j s_{ij} = 1$, $\boldsymbol{\sigma}_i = [\sigma_{ij}]_{1\leq j\leq N}$ and $\mathbf{s}_i = [s_{ij}]_{1\leq j\leq N}$ are similar to probability distributions. Thus divergences can be used to assess their mismatch. The cost function can be a sum of Kullback-Leibler divergences [13]:

$$E(\mathbf{X}; \boldsymbol{\Xi}, \mathbf{p}, \boldsymbol{\pi}) = \sum_i D_{\mathrm{KL}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) \ , \tag{2.3}$$

with

$$D_{\mathrm{KL}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) = \sum_j \sigma_{ij} \ln(\sigma_{ij}/s_{ij}) \ , \tag{2.4}$$

where $\mathbf{p} = [p_i]_{1\leq i\leq N}$ and $\boldsymbol{\pi} = [\mathbf{p}_i]_{1\leq i\leq N}$ contain the chosen precisions. This is the cost function of the basic SNE.

The cost function of NeRV blends two dual KL divergences. We call it KL of type 1:

$$D^{\kappa}_{\mathrm{KLt1}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) = (1-\kappa) D_{\mathrm{KL}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) + \kappa D_{\mathrm{KL}}(\mathbf{s}_i \| \boldsymbol{\sigma}_i) \ . \tag{2.5}$$

$0 \leq \kappa \leq 1$ is used to balances each term. $\kappa = 1/2$ give the type 1 symmetric generalization of the KL divergence $D^{1/2}_{\mathrm{KLt1}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i)$.

The cost function of Jensen-Shannon embedding (JSE) use KL of type 2:

$$D^{\kappa}_{\mathrm{KLt2}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) = \kappa D_{\mathrm{KL}}(\boldsymbol{\sigma}_i \| \mathbf{z}_i) + (1-\kappa) D_{\mathrm{KL}}(\mathbf{s}_i \| \mathbf{z}_i) \ , \tag{2.6}$$

where $\mathbf{z}_i = \kappa \boldsymbol{\sigma}_i + (1-\kappa)\mathbf{s}_i$. The cost function is written

$$E(\mathbf{X}; \boldsymbol{\Xi}, \mathbf{p}, \boldsymbol{\pi}, \kappa) = \frac{1}{\kappa(1-\kappa)} \sum_i D^{\kappa}_{\mathrm{KLt2}}(\boldsymbol{\sigma}_i \| \mathbf{s}_i) \ . \tag{2.7}$$

2.4 Optimization

The SNE-like methods have cost functions that are difficult to optimize with basic techniques like gradient descent. The cause may be the existence of many local minima and poor gradient scaling making the adjustment of the step size quite complicated. Thus the algorithm use optimization techniques based also on the second-order derivatives, like diagonal approximations of the Hessian, partial Hessian approaches of Quasi-Newton techniques. In this case we use the limited-memory version of the Broyden-Fletcher-Goldfarb-Shanno technique (L-BFGS). This technique is much faster than a basic gradient descent but it can also get stuck in shallow local minima.

2.5 Stochastic Neighbor Embedding

The similarities σ_{ij} and s_{ij} (Section 2.2.1 is seen in this algorithm as the probability for jth point to be part of the neighborhood of ith point in HD or LD respectively.

The aim of the basic stochastic neighbor embedding (SNE) algorithm [6] is to match these similarities between HD and LD by changing the LD coordinates. This is done by minimizing the sum of Kullback-Leibler (KL) divergences between σ_{ij} and s_{ij} (Equation 2.3).

The optimization technique depends on the implementation. We only consider, here, the basic gradient descent (which can be stuck in local minima) and the L-BFGS.

2.6 Neighborhood Retrieval Visualizer

Neighborhood retrieval visualizer (NeRV) [25] takes the problem in a different way but is nearly the same as SNE. It starts with the question "How to define the quality of a visualization of HD data". Its answer is to compare the neighborhood of each point. As for many other algorithm, we have false positives (FP), false negatives (FN) (or miss), true positive (TP) and true negative (TN) as we can see on Figure 2.2. Thus we can use classical performance measurements based

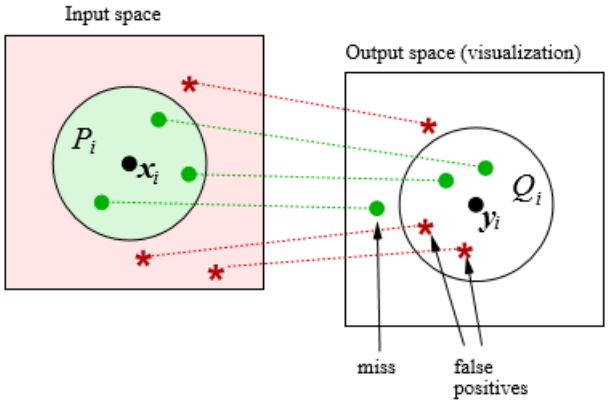


Figure 2.2: Diagram of the types of errors in visualization.

on precision P and the recall R:

$$P = \frac{TP}{TP + FP} \ , \tag{2.8}$$

$$R = \frac{TP}{TP + FN} \ . \tag{2.9}$$

But that interpretation is based on a binary observation (is in neighborhood or not). We can generalize that to a probability of being in the neighborhood which is the similarity.

The aim of the algorithm is also to minimize a cost function representing the precision and the recall of that interpretation. That cost function is the sum of dual KL divergences of type 1 (Equation 2.5).

2.7 t-distributed SNE

The t-distributed SNE (t-SNE) [24] is basically the same as the basic SNE except for two points.

Firstly, its cost function is a symmetrized version of the SNE one.

Secondly, its LD similarities are based on a Student-t distribution (a heavy-tailed distribution) instead of a Gaussian distribution. It allows to avoid the so-called crowding problem.

The crowding problem comes from the different scale of a hyper-shere in HD space or LD space. It scales as r^m where r is the radius and m is the dimension. Thus if we try to map in LD the HD neighborhood of a point, the area on the map will not be large enough.

2.8 Multi-scale SNE

The Multi-scale stochastic neighbor embedding (Ms. SNE) [12] computes a high-dimensional and a low-dimensional similarity for each pair of points in HD and LD.

Since the aim of the algorithm is to keep as well as possible the neighborhood structure of the data, it modifies the LD coordinates to reach the same similarities in HD and LD.

To do so, it minimizes a cost function (Section 2.3) using the limited-memory version of the Broyden-Fletcher-Goldfarb-Shanno technique (L-BFGS) and iterates over this process until a threshold is reached.

But the main particularity of this algorithm is the multi-scale approach. It does that search of best coordinates for multiple sizes of soft Gaussian neighborhoods to keep both local and global structures of the data and to ease the optimization (starting with global topology search and finishing with the local one)

2.9 Multi scale similarities

We have seen the possibility for a user to fix the neighborhood size to consider with perplexity $K_\star$. The multi-scale approach will compute similarities for different perplexities. Let us define the index h for them and π_{hi} and p_{hi} as the HD and LD precisions on scale h. We can rewrite the single-scale similarity formulas as

$$\sigma_{hij} = \frac{\exp(-\pi_{hi}\delta_{ij}/2)}{\sum_{k,k\neq i}\exp(-\pi_{hi}\delta_{ik}/2)} \quad \text{and} \quad s_{hij} = \frac{\exp(-p_{hi}d_{ij}/2)}{\sum_{k,k\neq i}\exp(-p_{hi}d_{ik}/2)} \tag{2.10}$$

The multi-scale similarities are defined as

$$\sigma_{ij} = \frac{1}{L}\sum_{h=L_{\min}}^{L_{\max}} \sigma_{hij} \quad \text{and} \quad s_{ij} = \frac{1}{L}\sum_{h=L_{\min}}^{L_{\max}} s_{hij} \tag{2.11}$$

L is the number of scales considered. The algorithm choice for $K_\star$ is to start with a small perplexity like $K_\star = 2$ and growing it with power of two. Thus we have $K_{h\star} = 2^{h-1}K_\star$ with $1 \leq L_{\min} \leq h \leq L_{\max}$. $L_{\max}$ doesn't exceed a value that will give a perplexity $K_{L_{\max}\star}$ bigger than N (number of points).

2.10 Perplexity dilemma in single scale optimization

The probability to observe local minima decreases with a higher value of perplexity. The larger size $K_\star$ is, the smaller precision π_i, the larger the bandwidths in the Gaussian similarities and the smoother the cost function. But using big values for the perplexity also reduces the aim of SNE-like methods which is to keep local neighborhoods. The choice is equivalent to choose between finding a globally good solution but locally wrong or get stuck in a local minimum but with a good local precision.

2.11 Multi-scale optimization

The multi-scale approach consists of computing the minimum of the cost function with a high value for the perplexity then reduces it and continues the search based on the first result. The target perplexity is $K_\star$ (2 in our case). We have $K_{h\star} = 2^{h-1}K_\star$, for $1 \leq h \leq \lfloor \log_2(N/K_\star) \rceil$. Thus the algorithm will start with $2^{\lfloor \log_2(N/K_\star) \rceil - 1}K_\star$ then divide the perplexity by two and so on until it reaches $K_\star$. The first LD coordinates to start the search with are given by keeping the P principal components of $\mathbf{\Xi}$.

2.12 Time complexity

Like MDS, the SNE-like method have to compare each pair of points. It is the principal bottleneck of these algorithms. If we have N points, that comparison is done in at least $O(N^2)$ time.

The multi-scale approach add a layer on the basic SNE methods. It does the optimization for $O(\log(N))$ perplexity levels. Thus we have a complexity of $O(N^2 \log(N))$.

The quadratic aspect of the time complexity of these algorithms is a real problem when we apply them on huge databases. It become very quickly impossible to compute the result in a reasonable amount of time.

In the next chapter, we will see some existing accelerated versions of single-scale SNE-like methods and their tools to achieve a reduction in the time complexity. Then we will see how we reduce the $O(N^2 \log(N))$ time complexity to a $O(N \log^2(N))$ one.

Chapter 3

Solutions to the Big Data problem

The Big Data field tries to revisit existing algorithms to make their computations in a short time on huge databases either by multi-threading or, generally with some approximations, by reducing their time complexity or a combination of both. Multi-threading can divide the amount of time by the number of threads (if they have the same computation power as the single thread initially used). But in some cases, it is not enough.

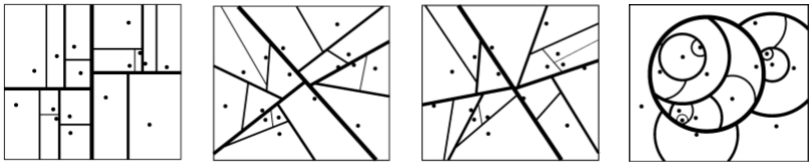
Let us illustrate the time complexity problem with an example. Suppose we have a database of N points and two algorithms A and B with a time complexity of $O(N)$ and $O(N^2)$ respectively. A and B do their computations, with $N = 1000$, in 10 minutes. Now they have to compute on a bigger database, $N = 100000$. A will take 100 minutes but B will do its work in 100000 minutes or approximately 70 days. Even if we multi-thread B the augmentation of time stays quadratic.

SNE methods are like algorithm B (or even worse in the case of Ms. SNE). We will briefly discuss about multi-thread and GPU calculation but we will mainly focus on the time complexity reduction. In the SNE field, the idea is to approximate the data with clusters and do a comparison between points and centroids. Multiple methods can be used to approximate the clusters. In this case we generally choose to use tree structures for two reasons. Firstly the clustering is done with a point of view: the concerned point. We need a different cluster set for all points. Secondly these clusters and their size depend on the distance between them and the point of view. Another possibility is to limit the number of interesting pairs, compares a point only with its neighborhood. It is also possible to decompose the problem using the FMM (Fast Multipole Methods).

3.1 Tree structure

The aim of the tree structure [10] is to divide the points and do a space-partitioning data structure. The structure obtained will keep the data topology and will be efficient to group points into clusters. Several techniques are possible (Figure 3.1), some of them are more useful in LD and some other in HD.

We will see in detail the K-dimensional tree and the Vantage Point tree.



(a) kd-Tree (b) PCA Tree (c) Ball Tree (d) vp-Tree

Figure 3.1: The partitioning of a 2D point set using different types of nearest neighbor trees, all with a maximum leaf size of 1 and a branching factor of 2. Line thickness denotes partition order (thicker lines were partitioned first). [10]

3.1.1 K-dimensional tree

The KDT (K-dimensional tree) is a binary tree. The non-leaf points generate a hyperplane that divides the space into two parts. Each child represent a half-space of the initial space divided by two. Each hyperplane is perpendicular to a particular axis (let us denote that axis x). Thus each point in the sub-tree that has a lower or equal (resp. higher) value for that x axis than the point representing the hyperplane will be on the left (resp. right) sub-tree. At the last level of the tree, there is only one point in each sub-space. The leafs are all the points.

The method to choose the dimension to divide is generally a simple cycle trough the axes. Suppose we have a three dimensional space (x, y, z), the root space will be divided regarding x, the two child will be divided regarding y then z then x and so on. The point representing a hyperplane is chosen by searching the one that will divide the most equally (in term of number of points) the concerned space. In other words, the point inserted in the tree is the median of the remaining points in the axis concerned.

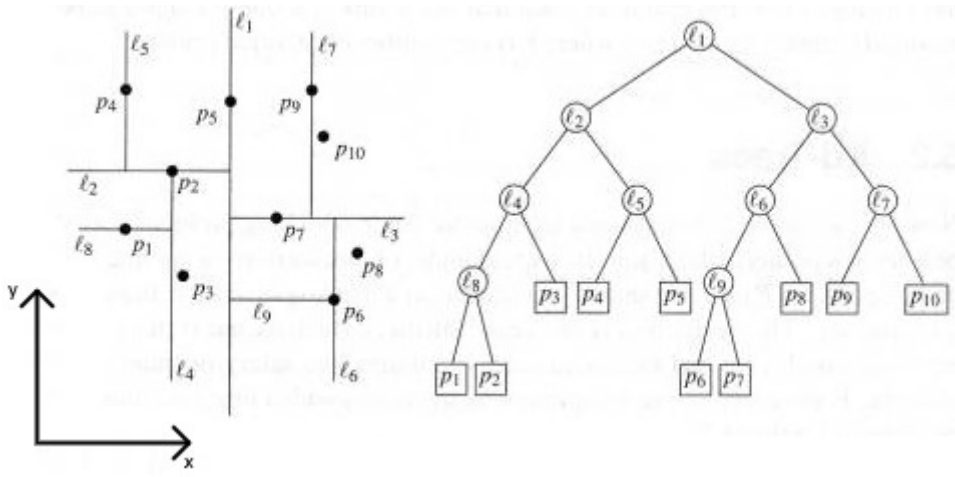


Figure 3.2: Example of a 2-D tree [1].

As an example (Figure 3.2), we have a 2-D space (x, y) with 10 points. The first axis chosen is x, the median point is p_5 and l_1 is the hyperplane perpendicular to x (a line in 2-D) created with p_5. l_1 become the root node. The left sub-tree contains p_{1-5} and the right one contains p_{6-10}. Then, with exactly the same process, p_2 and p_7 are chosen and so on.

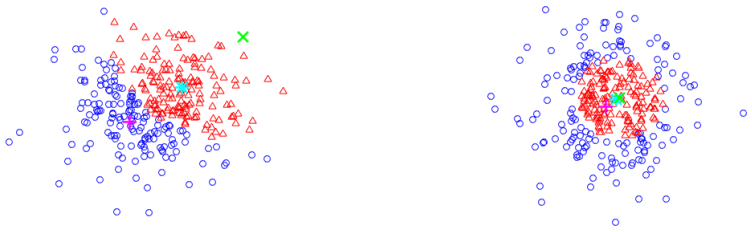
The time complexity of building the tree depends on the algorithm chosen to find the medians. The best is median of medians algorithm [3], it is a method based on the quickselect algorithm. It will choose a pivot then go trough the data to find the higher part and the lower part. Then we discard the part that does not contain the median and restart that process. The worst case of this algorithm is $O(N^2)$ (in the case where the chosen pivot is always the highest or the lowest value) but the mean time complexity is $\Omega(N)$. Whith such an algorithm, the time complexity is $O(N \log(N))$. A query in that tree takes $O(\log(N))$ time.

That tree is not suitable for high dimensional spaces. The relation between the number of points (N) and the number of dimension (K) have to be $N \gg 2^K$. A neighborhood can be defined with an hypersphere separating the data. The KDT partitions data regarding one dimension. Thus an hypersphere tends to intersect many adjacent sub-spaces. The search would become much slower [18].

3.1.2 Vantage point tree

The VPT (Vantage Point tree) is also a binary tree. It is similar to the KDT but instead of dividing the space with an hyperplane, it divides it with an hypersphere.

Firstly it chooses a point that will be the center of the hypersphere, called the VP (Vantage Point). Then it divides the space into two equal half-spaces (in term of number of points). The radius of the hypersphere is the distance between the VP and the median point. The space is divided in term of the absolute distance from the VP. The first half space contains half of the points that are the farthest from the VP and the second one contains the closest to the VP.



(a) VP is an external point (b) VP is an internal point

Figure 3.3: First division of the data using (a) an external point or (b) an internal point (like with a random selection). The green point is the VP, the magenta point is the ideal representative of the external cluster and the cyan point is the ideal representative of the internal cluster.

Several methods can be used to choose the VP, the two most used is the

random one and the most external point one. Random selection allows us to choose the VP in $O(1)$ but the most external point will divide better the space in our case. We need a representative for each half-space to represent a cluster. The aim achieved by choosing the most external point is to avoid to have wrong representatives (Figure 3.3). If the mean of the external points is inside the internal cluster, the chosen representative may be far away the real center of the cluster (Figure 3.3a).

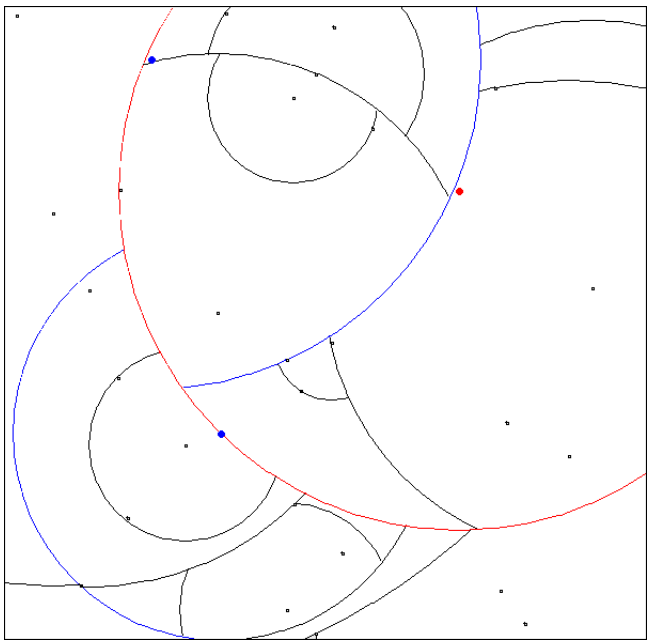


Figure 3.4: Example of a VPT on a 2-D space containing 30 points with a random choice of VP.

As an example (Figure 3.4), the first VP is chosen randomly, it is the red dot. The hypersphere (a circle in 2-D) is created using the median (corresponding to the metric of distance between the VP). The median is the blue dot on the red circle (which is the hypersphere separating the data in two parts). Then two others VPs are chosen (the two blue dots) ans so on.

The construction time complexity is $O(N \log(N))$ and the query is done in $O(\log(N))$ time [30].

Unlike KDT, VPT does not have to consider a quickly growing number of axes which make it better for high-dimensional searches.

3.1.3 Others

The following tree structure will not be used later in this paper. Thus we will explain them briefly for informational purpose.

PCA tree

The PCA (Principal Components Analysis) tree is similar to the KDT but it remedies the axis-alignment limitation by dividing the data with a hyperplane

perpendicular to the primary principal component. You can see the result of the PCA tree on Figure 3.1(b).

Ball tree

At each level of the Ball tree, a centroid of the concerned points is located. The center of the first child is the point with the greatest distance from that centroid. The second child's center is the point farthest from the first child's center. The division of the data is the hyperplane bisector of these two centers. A result can be seen on Figure 3.1(c).

3.2 Multi-threading: GPU and CPU

The aim of multi-threading is to divide the computation of an algorithm in several parts that will be computed at the same time. The basic technique is to re-factor the code into one or more big loop that does not depend on the previous iteration and send a part of that loop into each thread (Figure 3.5).

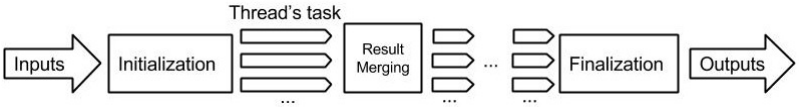


Figure 3.5: Structure of a paralleled algorithm

The CPU (Central Processing Unit) is less restrictive than the GPU (Graphical Processing Unit) since it allows to compute different algorithms on different threads. In this case, the problem becomes dividing the code in some independent parts. But we have generally less threads in a CPU (on average 8 threads with current CPU).

The GPU allows to launch the same computation on multiple data (SIMD) or multiple computation on multiple data (MIMD). With that more constrained model, it can launch a huge number of parallel computations [4].

3.3 Related works

Here are some concrete examples of existing accelerated version of SNE-like methods. Let us see how they put together the tools seen in previous sections.

3.3.1 Barnes-Hut SNE

The BHSNE (Barnes-Hut SNE) algorithm [23] uses many approximations. It uses a VPT (Section 3.1.2) in the high-dimensional space, a KDT (Section 3.1.1) in the low-dimensional space. It limits the number of relevant similarities in HD to $O(uN)$. And finally, it approximates the gradients of the cost function using the Barned-Hut algorithm.

Approximation of the HD similarities

The method builds a VPT with the HD data. That metric tree is then used to search for the neighborhood of each point. Then it computes the similarities only between a point and its neighborhood. Since similarity is a decreasing function of the distance between two points, the influence of a distant point is nearly equal to zero and can be neglected. Because of the tree approximation, a point i may be part of the neighborhood of a point j but j may not be in the neighborhood of i. A symmetrization process is done to keep the symmetric property of similarity. The resulted similarity formula is

$$\sigma_{j|i}^{BH} = \begin{cases} \sigma_{ij}, & \text{if } j \in \mathcal{N}_i \text{ (Eqn. 2.1)} \\ 0, & \text{otherwise} \end{cases} \tag{3.1}$$

$$\sigma_{ij}^{BH} = \frac{\sigma_{j|i}^{BH} + \sigma_{i|j}^{BH}}{2N} \ , \tag{3.2}$$

where $\mathcal{N}_i$ is the neighborhood of the point i. By doing that, they have a sparse matrix where all zeros do not have to be computed.

Approximation of the Gradients

The algorithm is based on the t-SNE gradient. It rewrites it as follows:

$$\frac{\delta C}{\delta x_i} = 4(F_{attr} - F_{rep}) = 4\left(\sum_{j \neq i} \sigma_{ij}^{BH} s_{ij}^{BH} Z(x_i - x_j) - \sum_{j \neq i} (s_{ij}^{BH})^2 Z(x_i - x_j)\right) \ , \tag{3.3}$$

where $Z = \sum_{k \neq l} (1 + \|x_k - x_l\|^2)^{-1}$. F_{attr} is already fast to compute with a sparse matrix of σ_{ij}^{BH} due to the neglected distant points. But there is still N^2 similarities for s_{ij}^{BH}. Suppose we have two points close to each other x_j and x_k and one distant from both of them x_i, we have $\|x_i - x_j\| \approx \|x_i - x_k\| \gg \|x_j - x_k\|$. The contribution of x_j and x_k will be nearly the same in F_{rep}. The algorithm will build a KDT (Section 3.1.1) (Called Quadtree in the Barnes-Hut SNE paper) in LD and create a cluster set, where each cluster is represented by an existent point, by searching in that tree with a depth-first search. Then we can approximate F_{rep} by regrouping s_{ij}^{BH} together for all point j close to each other. The contribution becomes $N_{cell}(s_{i,cell}^{BH})^2 Z(x_i - x_{cell})$ where $(s_{i,cell}^{BH})^2 Z = (1 + \|x_i - x_{cell}\|^2)^{-1}$.

The resulted time complexity is $O(N \log(N))$ instead of $O(N^2)$ for a very small loss of precision (Figure 3.6).

3.3.2 Fast Multipole Methods

The idea of FMM (Fast Multipole Methods) [27] is to do a series expansion of the Gaussian similarities in order to decompose each similarity into each term of the series. Truncate these series to terms of up to order p transforms the time complexity $O(N^2)$ to $O(Np^d)$ in d dimensions (Not expensive in our case since the aim is the visualization of data and so $d \leq 3$). The parameter p can be chosen by the user.

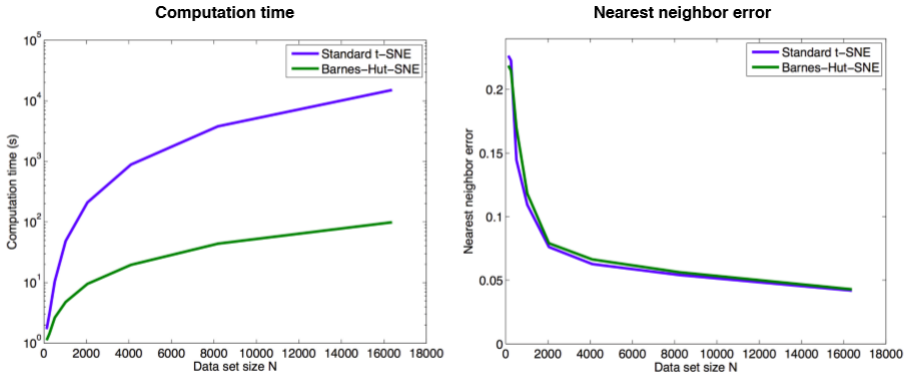


Figure 3.6: Computation time (in seconds) required to embed MNIST digits (left) and the 1-nearest neighbor errors of the corresponding embeddings (right) as a function of data set size N for both standard t-SNE and Barnes-Hut-SNE. Note that the required computation time is plotted on a logarithmic scale. (Figure 3 : [23])

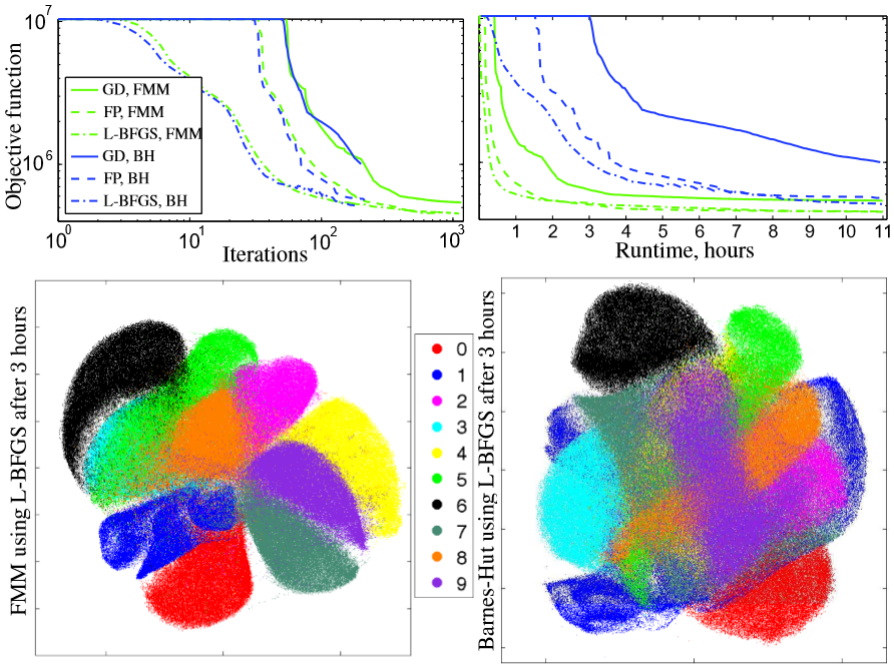


Figure 3.7: The embeddings of 1020000 digits from MNIST dataset with FMM and Barnes-Hut (BH) using gradient descent (GD), fixed-point iteration (FP) and L-BFGS. Top: the objective value change with respect to the number of iterations and the runtime. Bottom: the embedding of FMM and BH with L-BFGS after 3 hours of optimization. (Figure 1 : [27])

If we fix p and d, the algorithm has a linear time complexity. Regarding the BH SNE, the FMM is much faster (Figure 3.7).

Chapter 4

Accelerated Ms. SNE

4.1 Main concept of the solution

As a reminder, the original Ms. SNE (Multi-scale Stochastic Neighbor Embedding) is composed of three parts. Firstly, it initializes the LD data with a PCA and computes the HD similarities, the perplexity levels and the precisions. Then for each perplexity level, it computes, with at most 25 iterations of the L-BFGS optimization, the new LD coordinates to have a value as small as possible for the chosen cost function (KL, KL type 1, KL type 2).

The accelerated version computes a VPT for the HD similarities. It uses a new data structure called TSHILT for the LD similarities and does not use anymore the L-BFGS for the optimization part but uses an approximation of the gradient descent instead.

4.2 Data structures

We have already defined most of the tools that will be used in the solution (Chapter 3). We will now see how to use them in this algorithm.

4.2.1 Asymmetric distance matrix

The original Ms. SNE compares each pair of points using a $N*N$ distance matrix and computes N^2 similarities. The mere existence of that $N * N$ matrix makes the time (and space) complexity quadratic. The accelerated version partitions the data in a different way for each point and obtain a variable number of clusters.

Each cluster is represented by the closest point to the center of the cluster. Thus we need to store the index of the representative and the number of points contained in each cluster. To store that information we use two vectors of cells. Each cell contains a variable number of item depending on the point of view. Our space complexity is reduced to $O(N \log(N))$.

All the code has been adapted to that new data structure.

4.2.2 Query on tree

To retrieve the clusters we search the point of view (the concerned point that will be compared to the clusters for the similarities computation) in a tree structure. The first idea is at each level we take the child of the sub-space that does not contained the point of view and store a cluster representing this sub-space then continue on the next level.

With such a cluster selection, we obtain bigger clusters far away the point of view and smaller ones close to it. The big clusters are found at the top of the tree and the lower ones at the bottom of the tree. We ensure to take in account all points because no subspace is unexplored or used to make a cluster.

We have implemented the possibility for the user to choose a precision level for the query algorithm. Suppose we have a precision level of i, it means that, instead of taking the first subspace encountered that does not contained the concerned point as a cluster, we continue on the sub-tree for at most i level and make clusters for the at most 2^i children.

It allows us to create more clusters of a smaller size. Thus the points in each cluster are on average closer than with a null precision level. It makes the approximation closer to the reality.

4.2.3 VPT for HD similarity

As we saw in Section 3.1.2, the VPT is more suitable in high dimension, we have chosen to use that one because of the good results of the other accelerated version of SNE-like methods. We use the most external point of the subspace to be the vantage point.

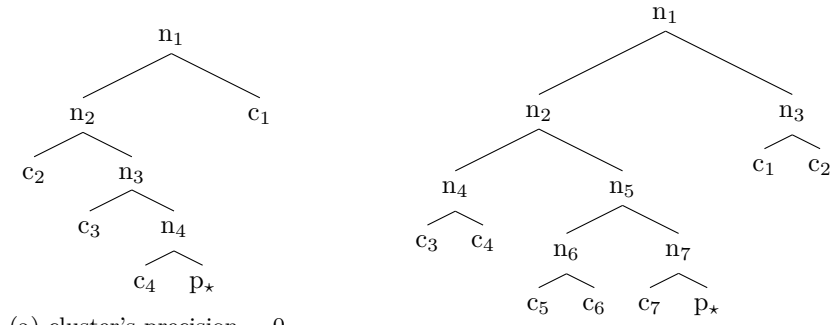
In the BH SNE, they only use the VPT to select the local neighborhood ($K \ll N$) of a point. In our case we use it to approximate all points with clusters. It is necessary to consider all neighborhood sizes and not only the small neighborhood ones.

Concretely, for each point we choose clusters of different sizes, big cluster far away from the concerned point and smaller clusters for the closer areas. To do that, we need a point representative of a subspace that will be used as centroid of a cluster. Thus, for each subspace created during the construction of the VPT, we search the closest point to the center of the points located in the subspace. We also need the number of points remaining in the subspace to weight all our computation.

That information is computed in $O(N_{subspace})$ time. Thus the global time complexity of the construction remains $O(N \log(N))$.

The search query is computed with the distance between the VP and the point of view. If it is greater (resp. lower) than the hypersphere radius, we continue the search in the external (resp. internal) subspace and make cluster(s) with the internal (resp. external) one. Here is an example of the result of a query on a small data-set (Figure 4.1).

The time complexity of the query is $O(\log(N))$ for a fixed cluster's precision P. There are, on average, two times more clusters by precision level. The global time complexity of a query is $O(2^P \log(N))$ where P is the cluster's precision. Since small values for P are suitable, that exponential part is not a problem.



(a) cluster's precision = 0

(b) cluster's precision = 1

Figure 4.1: Query example for VPT or KDT of the clusters of the point $p_\star$ where n_x are nodes and c_x are clusters (or points). Note that c_4 in (a) is a point as for c_5, c_6 and c_7 in (b)

4.2.4 KDT for LD similarity

We first chose to use KDT for the LD similarities. As for the HD ones, the idea was to do clustering for each point. But this solution, in our case, gives us poor results. The convergence of the optimization part was slow and 25 iterations were not enough at all. The reason is that the KDT is only based on LD. The clusters chosen by the search query do a good partitioning of the data in LD. But in these clusters, we find very distant points if we look at the HD coordinates.

We call the homogeneity of a cluster its property to have only points close to each other. In the case of the KDT, the homogeneity is guaranteed in LD but not at all in HD. To find more useful clusters, we need to add HD information in the structure.

The search query was like the VPT one but instead of looking at the distance, we looked at the dimension used to separate data at the concerned level of the tree.

4.2.5 Twin Spaces Hybrid Inter-Leaved Tree

We develop the TSHILT (Twin Spaces Hybrid Inter-Leaved Tree) to add some information to help the optimization to converge faster.

The basic idea is to merge the LD KDT with the HD VPT. The odd levels of the tree are constructed based on LD data with a KDT. The even levels are built with the HD data with a VPT. The presence of HD division of the data gives additional information than the KDT.

The LD homogeneity of the clusters is guaranteed by the LD separations levels of the tree. Unlike the KDT, the HD homogeneity is also guaranteed by the HD separations levels. Since we have a better HD separation, the algorithm will be able to separate distant points in HD that are close in LD faster.

The construction of the tree is done in $O(N \log(N))$ time since it uses the same operations as KDT or VPT and it has the same size.

The query is a little bit more complicated than in the two other cases. Since we try to make big clusters far away from the concerned point and small clusters

close to that point in term of the LD distance, we have to explore both children at each HD level. Let us show you an example with a cluster's precision of 1 (Figure 4.2).

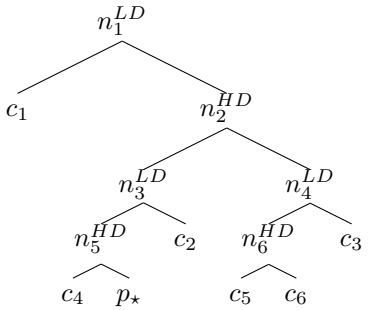


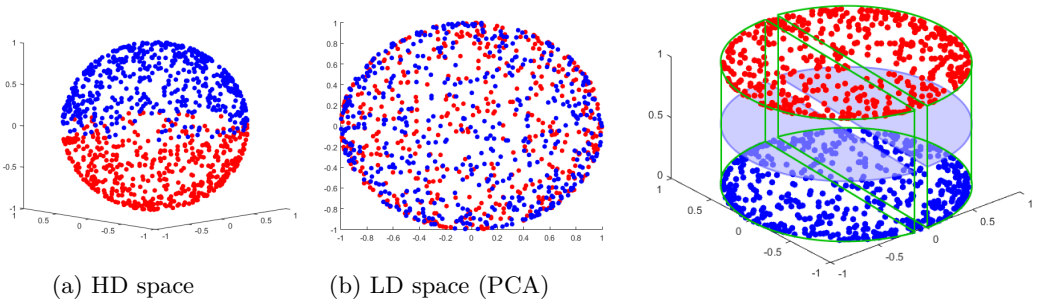
Figure 4.2: Query example for TSHILT of the clusters of the point $p_\star$ where n_x^{HD} (resp. n_x^{LD}) are nodes on the HD (resp. LD) levels and c_x are clusters (or points).

The query seems a little bit heavy concerning the time complexity, let us analyze the time complexity of that query. There are $O(\log_2(N))$ levels in the tree. So we have $O(\log_2(N))/2$ HD levels. For each HD level, we double the search. The number of visited nodes is

$$1+2+2+4+4+...+2*2^{O(\log_2(N))/2} = 1+2+2+...+O(N) \Rightarrow O(\log(N)) \quad (4.1)$$

We can conclude the total time complexity to be $O(\log(N))$.

We can see an illustration of the data partitioning done by TSHILT on Figure 4.3.



(a) HD space (b) LD space (PCA)

(c) TSHILT illustration

Figure 4.3: Illustration of the data partitioning done by TSHILT. The green volumes are the two first subspaces obtained by a LD division. The two blue plan are the separation done in HD

4.3 Challenges

4.3.1 Compare HD and LD similarities

Since the data is not the same in HD and LD, even the data structures are not the same, the first problem was to compare the same thing. The clusters selected in both dimensional systems are different.

SNE-like methods try to adjust LD coordinates to have the same similarities in LD as in HD. Thus we have to compare similarities that represent the same data. To achieve this aim, we compute the HD and LD similarities with the HD clusters and the LD clusters.

4.3.2 Gradient descent instead of L-BFGS

In the original algorithm, L-BFGS is used to find the exact minimum in the cost function evaluation. But the method supposes that the result of the next step is always better than the result in the previous step. Unfortunately, since we change the clusters between two iterations, it is not always the case. The clusters define the similarities which define the cost function evaluation. The LD clusters change. Thus the meaning of the similarities also changes. The accelerated version approximates the true cost function. As a result of noise in the evaluation, it is impossible to tell for sure whether a lower value of the cost function indicates real progress in the minimization or is just an artifact of the tree structure.

We do a basic gradient descent instead of the BFGS. We may miss the optimal solution but it allows to the cost function value to be worse than the previous iteration.

4.3.3 Convergence of optimization

Without TSHILT, thus with a data structure only based on the LD coordinates, the algorithm had difficulties to converge to an optimal solution in some cases. That can be explained with the initialization. The initialization is a PCA, it takes the K first orthogonal dimensions combination that has the biggest variance. Doing this is equivalent to simply erase some information.

For example, in the case of a random generated 3D sphere, keeping 2 dimension is equivalent to crush that sphere and to erase the information of which hemisphere each point belongs to. In that example, the algorithm makes cluster with points of both hemispheres. At each iteration it builds different clusters and in average, the disc keep his form instead of taking the shape of an unfolded sphere.

That's why TSHILT is very useful. At each HD level, the separation of the data keep the information missed in LD.

4.3.4 Symmetrization of the gradient and Hessian weight matrices

We need a symmetrization because of the asymmetry of similarity. Euclidean distance is symmetric, the non-normalized similarity too but when we normalize it, it loss that property.

Function	Comp. time Matlab (s)	Comp. time MEX (s)
binasimi	724	451
binaquery	424	11
tshiltquery	1927	33
Acc. Ms. SNE	3476	868

Table 4.1: Time execution comparison of the query functions and the similarities computations with and without micro optimization for the reduction of a 3-D sphere of 2000 points in 2 dimensions

The cost function evaluation (CFE) computes the gradient and approximates the diagonal of the Hessian. To achieve that, it computes weights that have to be symmetric as well. But a symmetric matrix would have a size of $N * N$ and we will get back to a quadratic time complexity. The challenge is to make our vector of cells symmetric.

It is impossible, thus let us take a more abstract concept. The symmetrization is done by adding to the matrix his transpose. If we can get a kind of transposed vector of cells of the same size than our data structure, we could have the symmetric structure.

We could write the similarity between the point p_i and p_j as $s_{ij} = S_{ij}/n_i$ and between p_j and p_i as $s_{ji} = S_{ji}/n_j = S_{ij}/n_j$ where S is the symmetric part of the similarity and n is the normalization part. The transpose of s_{ij} is written as Sij/n_j.

Each cell represents all similarities of a point. In a cell, each item represents a cluster. A cluster is represented by an existing point. Suppose we apply all computations done on the row i on the item of a cluster represented by p_i. It is exactly the content of the transposed matrix.

In the code, we do all computations done on a row on all items represented by the concerned point. At the end we add the two matrix together and we obtain the "symmetric" matrix.

4.3.5 Micro optimizations

Even if the accelerated method has an improved time complexity ($O(N \log^2(N))$) regarding the time complexity of the original algorithm ($O(N^2 \log(N))$), it has also some part that does not change the worst case time complexity. Which result in a bigger cost in term of computation time for smaller data sets. To obtain a better computation time for the accelerated version than the original method, we had to test the algorithms on a big database which took a lot of time. To be able to generate good results we had to do some micro optimizations.

The code has been developed on Matlab which does a lot of check for matrix computation. A solution to avoid some useless check is to re-code the inefficient and time-consuming functions in C using MEX-functions. This has been done for all the query on trees and for the computation of the similarities (tshiltquery, binaquery and binasimi).

As you can see (Table 4.1), passing these matlab function into MEX-functions was really effective and allows us to have good results faster.

Chapter 5

Results

Before describing our results, we need to define the quality assessment method [15, 26]. It is not an easy part to quantify "Is this a good representation of the HD data".

5.1 Quality assessment: Rank-based criteria

5.1.1 Ranks

The ranks (ρ_{ij} in HD space and r_{ij} in LD space) are used to quantify the neighborhood of the points. It orders for a point i all points from the closest to the farthest. In other words, a rank ρ_{ij} computes the cardinality of the set of neighborhood of the point i until the point j (Figure 5.1).

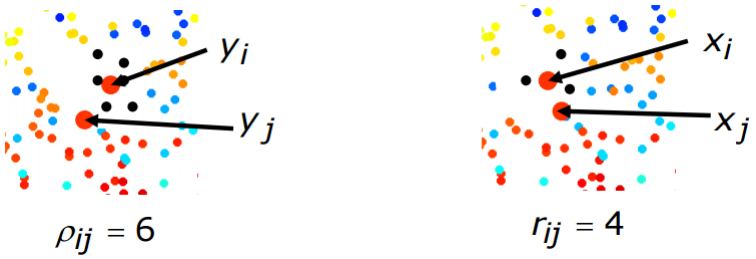


Figure 5.1: Illustration of the ranking computation method. Black points are closer to y_i (resp. x_i) than y_j (resp. x_j) in HD (resp. LD) (Slide 96: [26]).

ρ_{ij} and r_{ij} are the metric of the quality assessment method.

5.1.2 Co-ranking matrix

The co-ranking matrix $Q = [q_{kl}]_{1 \leq k,l \leq N-1}$ organizes the ranks for all pairs of points. Each item q_{kl} contains the number of pair of points (i, j) that have $\rho_{ij} = k$ and $r_{ij} = l$. We can write it as

$$q_{kl} = |\{(i,j) | \rho_{ij} = k \text{ and } r_{ij} = l\}| \tag{5.1}$$

Let us define a variable $1 \leq K \leq N$, it defines blocks in the co-ranking matrix (Figure 5.2). It also defines to which K-neighborhood the algorithm measures the quality. A perfect solution would have $\sum_i q_{ii} = N^2$ and $\sum_{ij,i\neq j} q_{ij} = 0$. We define an extrusion (resp. intrusion) error as a point that leaves (resp. enter in) a neighborhood between the HD and LD space.

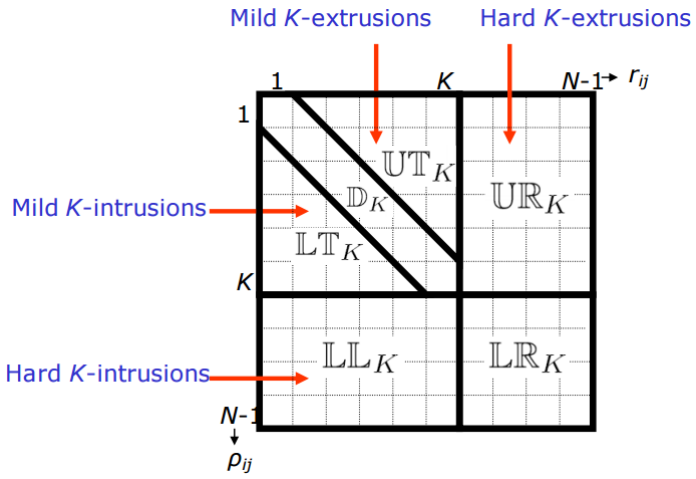


Figure 5.2: Illustration of the co-ranking matrix blocks (Slide 100: [26]).

We define $\mathbb{F}_K = \{1, ..., K\}$ and $\mathbb{L}_K = \{K+1, ..., N-1\}$. We can define all blocks as

$$\mathbb{UR}_K = \mathbb{F}_k \times \mathbb{L}_k \ , \tag{5.2}$$
$$\mathbb{D}_k = \{(i,i) : 1 \leq i \leq K\} \ , \tag{5.3}$$
$$\mathbb{LL}_k = \mathbb{L}_k \times \mathbb{F}_k \ , \tag{5.4}$$
$$\mathbb{LT}_k = \{(i,j) : 1 < i \leq K \text{ and } j < i\} \ , \tag{5.5}$$
$$\mathbb{LR}_k = \mathbb{L}_k \times \mathbb{L}_k \ , \tag{5.6}$$
$$\mathbb{UT}_k = \{(i,j) : 1 \leq i < K \text{ and } j > i\} \ . \tag{5.7}$$

5.1.3 Quality assessment computation

The K-ary neighborhoods of $\boldsymbol{\xi}_i$ and $\mathbf{x}_i$ are the sets defined by $v_i^K = \{j : 1 \leq \rho_{ij} \leq K\}$ in HD and $n_i^K = \{j : 1 \leq r_{ij} \leq K\}$ in LD. We can write the K-ary neighborhood preservation as

$$Q_{NX}(K) = \frac{1}{KN} \sum_{i=1}^{N} |v_i^K \cap n_i^K| \ , \tag{5.8}$$

with $1 \leq K \leq N-2$. We can rescale that criterion as

$$R_{NX}(K) = \frac{(N-1)Q_{NX}(K) - K}{N-1-K} \ . \tag{5.9}$$

$R_{NX}(K)$ has a range from 0 (random embedding) to 1 (perfect embedding). A scalar value to quantify the quality is the AUC (Area Under the Curve) of that criterion along all K (algorithmic scaled). We write it as

$$AUC_{ln(K)}(R_{NX}(K)) = \frac{\sum_{K=1}^{N-2} R_{NX}(K)/K}{\sum_{K=1}^{N-2} 1/K} \tag{5.10}$$

That is the measure we will use to assess the quality of the embedding. The higher the value is, the better it is.

5.2 Computation time comparison

The theoretical time complexity of Ms. SNE and Acc. Ms. SNE are respectively $O(N^2 \log(N))$ and $O(N \log^2(N))$. To verify these properties, we have measured the computation time of both algorithms for differents number of random points on a 3D sphere (Figure 5.3).

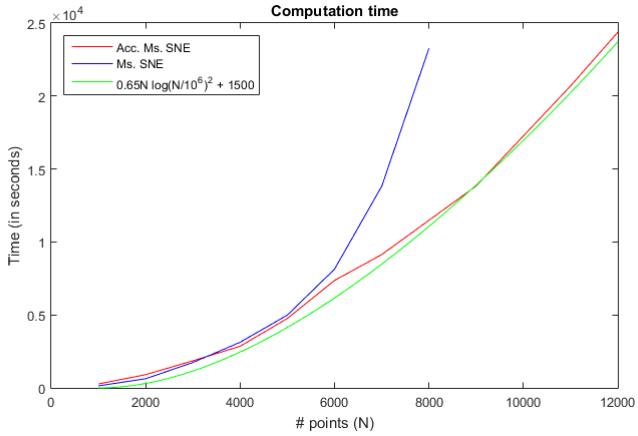


Figure 5.3: Illustration of the time complexity with a computation time comparison of MS. SNE and Accelerated Ms. SNE.

We can easily observe an improvement between the two methods. Moreover, we can approximate the computation time of the Acc. Ms. SNE with the function $0.65N \log^2(N/10^6) + 1500$.

We stopped the computation at $N = 8000$ for the original method because of its quadratic complexity.

We have also observed during the multiples runs that the accelerated version computation time vary a lot for a high number of points. That can be explained with the randomness of the data used. Some data can be summarized more efficiently with the tree structures. In this case we will have less clusters. Let us denote the number of clusters in average as C. Then the time complexity can also be written as $O(NC \log(N))$. Which explains the difference of computation time with the same number of points but different databases.

5.3 Visual comparison

The first quality assessment is visual. We have generated results on three differents databases and label them with a color map (Figure 5.4). The three data sets are the random 3D sphere, the MNIST database [11] (1000 points) and the Coil database [19] (1440 points).

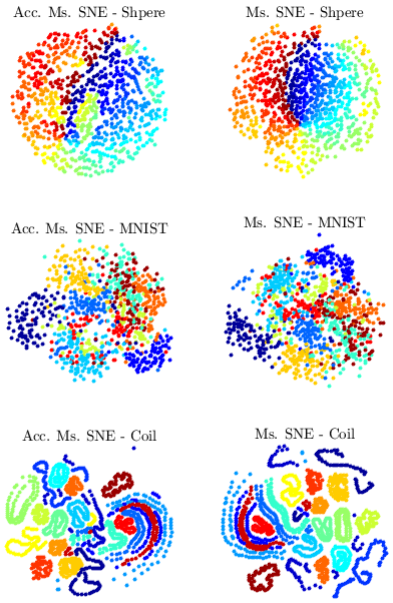


Figure 5.4: Comparison of the results given by Acc. Ms. SNE and Ms. SNE on differents databases using the classical KL as cost function

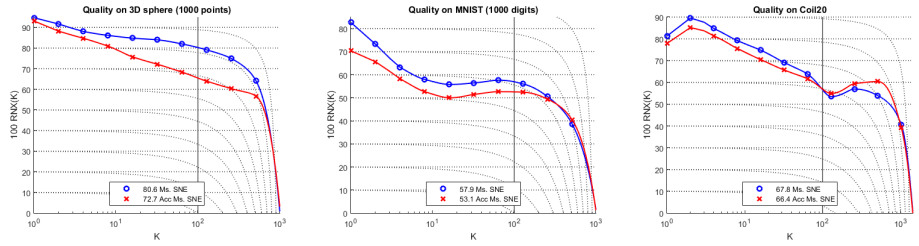
The result of both methods are very similar. We can observe on the sphere case that there are some mistakes. Because of all our approximation, the accelerated methods is stuck in a local minima.

5.4 $AUC_{ln(K)}(R_{NX}(K))$ comparison

A quantification of the quality assessment is done with the concepts seen in Section 5.1. On these graphs (Figure 5.5), we can observe the quality for the local (resp. global) structure (small (resp. large) neighborhood) for small (resp. high) value of K.

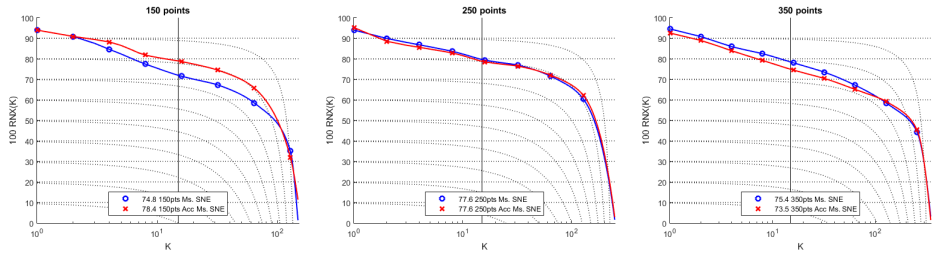
For the sphere data set, the local structure is well conserved but because of the local minima, the global structure have more loss of precision. The two other data set give a similar precision but the accelerated version is globally less precise than the original method due to the approximations use in the computation.

We can also analyze an interesting fact for small number of points (Figure 5.6).



(a) Sphere data set (b) MNIST data set (c) Coil data set

Figure 5.5: Quality assessment of Acc. Ms. SNE (in red) and Ms. SNE (in blue) on differents databases



(a) 150 points (b) 250 points (c) 350 points

Figure 5.6: Quality assessment of Acc. Ms. SNE (in red) and Ms. SNE (in blue) on differents small number of points in the case of the sphere database.

The result of the accelerated (and normally approximated) version is better than the original version for very small number of points. That can be explained by the number of clusters used by the algorithm. For these numbers of points, the algorithm computes similarities with more clusters than the number of points. This high number of clusters comes from the HD clusters and the LD clusters since we keep all of them to compute the HD similarities and LD ones.

Chapter 6

Conclusion

The Ms. SNE is at the cutting edge in the dimension reduction field. We can conclude with this thesis that its accelerated version is possible and is on the right track. Despite of the loss of precision, we have proved that the algorithm has a better time complexity.

The TSHILT data structure has proven its efficiency. This is a quite original data structure. It would be interesting to integrate it in other algorithms to evaluate its contribution to the convergence rapidity in the optimization part of SNE-like methods. We could also maybe integrate it in the accelerated methods of quality assessment.

There are still work to do but, once it will be done, the Acc. Ms. SNE could be the best existing algorithm in the DR field for big data. Thanks to the $O(N \log(N^2))$ time complexity, for large data we have a nearly linear time complexity. With some parallelizations (like GPU computation) we could flatten the computation time curve of Figure 5.3.

This master thesis comes to an end but not the project! This work was really interesting and I would like to continue and improve the code to obtain a really competitive one. To do so, here are some possibles improvements.

6.1 Explore more solutions

6.1.1 FMM

We see some results of the efficiency of FMM in Section 3.3.2. It is a way more complicated acceleration technique but since it has reduced a $O(N^2)$ time complexity into $O(N)$, we may implement it in the Ms. SNE to obtain a $O(N \log(N))$ time complexity.

6.1.2 Tree structure comparison

As seen in the Section 3.1, There are a lot of different tree algorithms. We base our decisions on the results of other works but to confirm that choice, we have to test other data structures and compare the efficiency. Maybe we can also improve TSHILT which is a quite original technique for partitioning data.

6.1.3 Other optimization methods

Because of a lack of convergence speed, we drop the idea of using L-BFGS but the problem could be solved in another way. The current optimization method is not the best at all. It is a fact, that part causes a loss of precision in the final result.

There are many algorithms that could be used to do the optimization part. As for the other tree structures, we have to test some of them and compare the efficiency.

6.2 Micro optimization

Once we have good choices concerning the previous points, another improvement can be done, the micro optimization.

6.2.1 Grid search

We fix some parameters in the accelerated algorithm (for example the precision of the query in the trees) and there will certainly be more of them in the final version. We have to give the possibility to the user to choose these variables and to choose for him the bests as possible default values. For the last point, we need to do a grid search of these parameters that may interact between each other.

6.2.2 GPU compatibility

The original Ms. SNE algorithm is GPU compatible. But because of the new code, the trees, the asymmetric distance matrix, etc, the accelerated version is not anymore GPU compatible. That feature will allow us to do visualization on even bigger databases.

6.2.3 Code re-factoring

Finally, this is a research code, the documentation is not good enough, the utilization is not intuitive. The algorithm need to be re-factored, well documented and commented. Maybe developed on multiple platforms to maximize its utility. In other words, it is a prototype.

In development mode, it is useless to do these jobs since the code will change to become better and better. Once in production mode, we have to think about the user.

Bibliography

[1] Kd-Tree applet demo page. http://homes.ieu.edu.tr/hakcan/projects/kdtree/kdTree.html. Accessed: 2015-08-09.

[2] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 2001)*, volume 14. MIT Press, 2002.

[3] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.

[4] Ralph Duncan. A survey of parallel computer architectures. *Computer*, 23(2):5–16, 1990.

[5] Imola K Fodor. A survey of dimension reduction techniques, 2002.

[6] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 833–840, 2002.

[7] A.J. Kearsley, R.A. Tapia, and M.W. Trosset. The solution of the metric STRESS and SSTRESS problems in multidimensional scaling using newton's method. *Computational Statistics*, 13(3):369–396, 1998.

[8] Reinhard Koch and Fay Huang. *Computer Vision–ACCV 2010 Workshops: ACCV 2010 International Workshops. Queenstown, New Zealand, November 8-9, 2010. Revised Selected Papers*, volume 6468. Springer, 2011.

[9] T. Kohonen. Self-organization of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[10] Neeraj Kumar, Li Zhang, and Shree Nayar. What is a good nearest neighbors algorithm for finding similar patches in images? In *Computer Vision–ECCV 2008*, pages 364–378. Springer, 2008.

[11] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[12] John A Lee, Diego H Peluffo-Ordóñez, and Michel Verleysen. Multi-scale similarities in stochastic neighbour embedding: Reducing dimensionality while preserving both local and global structure. *Neurocomputing*, 2015.

[13] John A Lee, Emilie Renard, Guillaume Bernard, Pierre Dupont, and Michel Verleysen. Type 1 and 2 mixtures of kullback–leibler divergences as cost functions in dimensionality reduction based on similarity preservation. *Neurocomputing*, 112:92–108, 2013.

[14] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[15] John A Lee and Michel Verleysen. Quality assessment of dimensionality reduction: Rank-based criteria. *Neurocomputing*, 72(7):1431–1443, 2009.

[16] John A Lee and Michel Verleysen. Shift-invariant similarities circumvent distance concentration in stochastic neighbor embedding and variants. *Procedia Computer Science*, 4:538–547, 2011.

[17] John Aldo Lee, Michel Verleysen, et al. Unsupervised dimensionality reduction: from principal component analysis to modern nonlinear techniques. In *43e Journées de Statistiques (JDS 2011)*, 2011.

[18] Sameer Nene, Shree K Nayar, et al. A simple algorithm for nearest neighbor search in high dimensions. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 19(9):989–1003, 1997.

[19] Sameer A Nene, Shree K Nayar, Hiroshi Murase, et al. Columbia object image library (coil-20). Technical report, Technical Report CUCS-005-96, 1996.

[20] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.

[21] J.W. Sammon. A nonlinear mapping algorithm for data structure analysis. *IEEE Transactions on Computers*, CC-18(5):401–409, 1969.

[22] J.B. Tenenbaum. Mapping a manifold of perceptual observations. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems (NIPS 1997)*, volume 10, pages 682–688. MIT Press, Cambridge, MA, 1998.

[23] Laurens van der Maaten. Barnes-hut-sne. *arXiv preprint arXiv:1301.3342*, 2013.

[24] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

[25] Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *The Journal of Machine Learning Research*, 11:451–490, 2010.

[26] Michel Verleysen. Lecture notes in machine learning (lelec2870): chapter nonlinear dimensionality reduction, 2014.

[27] Max Vladymyrov and Miguel A Carreira-Perpinán. Linear-time training of nonlinear low-dimensional embeddings. In *Proceedings of AISTATS 2014, International Conference on Artificial Intelligence and Statistics, JMLR W&CP*, volume 33, 2014.

[28] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006.

[29] G. Young and A.S. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22, 1938.

[30] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

List of Figures